Analysis of the Multiple Excess-S Modulo K (MSK) Coding Scheme

Nadine Fea
Computer Science Department
Canterbury University
Christchurch, New Zealand
nef13@cosc.canterbury.ac.nz

Supervisors: Malcolm Shore and Ray Hunt

Contents

List of Figures

List of Tables

Abstract

The coding scheme known as MSK, an extension to an early commercial Excess-3 coding scheme, has been proposed by ML and S Keshariya as a cryptographic function suitable for improving the strength of stream ciphers. The authors make the claim that the scheme "exponentially enhances" the security of stream ciphers. Four schemes for applying MSK are proposed, from the simplest form whereby MSK is applied to the keystream, to the most complex incorporating two MSK modules and a non-linear combining function. This claim is investigated through empirical testing. The formal test regimes available for assessing randomness are considered and the Monobit, Maurer's Universal and the Approximate Entropy tests are selected to determine the extent of improvement achieved by using MSK. A set of arbitrary pseudorandom number generators are created based on linear feedback shift registers and these are used for initial verification of the MSK claim. The GSM system's A5/1 algorithm is also used for verification of real world usefulness of MSK. The results in this the report will show that the author's claim cannot be sustained and that the MSK coding scheme does not make any significant difference to the security, that is an improvement in randomness, of a stream cipher.

1 Introduction

This report will begin with a review of stream ciphers, covering the various forms of stream ciphers in use and giving examples of actual algorithms including the A5 algorithm. The stream cipher review will the become more focused on a particular class of stream ciphers based on linear feedback shift registers (LFSRs) and will look at how such mechanisms can be used to create stream ciphers.

A critical aspect of stream ciphers is their randomness. This report will also review the methods of testing randomness and in particular the suite of statistical tests provided by NIST Special Publication 800-22.

The A5 stream cipher algorithm is the keystream generator used for the experiments in this report. The General System Mobile (GSM), more commonly called the Global System for Mobiles, uses the A5 algorithm for its traffic encryption. The A5 algorithm has been the subject of significant criticism from the academic cryptanalytic community.

The MSK coding scheme has been recently published and the authors promote it as a means for "exponentially improving" the security of stream ciphers. As yet this algorithm has not been proved. This report describes the application of MSK to the A5 stream cipher and assesses whether this claim is valid.

2 Stream Ciphers

There are two general types of key-based algorithms: symmetric and public-key. Symmetric algorithms, sometimes called conventional algorithms, are algorithms where the encryption key can be calculated from the decryption key and vise versa. In most symmetric algorithms, the encryption key and the decryption key are the same. These algorithms require that the sender and the receiver agree on a key before they can communicate securely.

Symmetric algorithms can be divided into two categories; stream ciphers operate on the plaintext or single bit (or sometimes byte) at a time and block ciphers operate on plaintext in groups of bits or blocks. For most computer algorithms a typical block size is 64 bits, large enough to preclude analysis and small enough to be workable [29]. With a block cipher, the same plaintext block will always encrypt to the same ciphertext block, using the same key. With a stream cipher, the same plaintext bit or byte will encrypt to a different bit or byte every time it is encrypted.

Stream ciphers can be designed to be exceptionally fast, much faster than any block cipher. While block ciphers operate on large blocks of data, stream ciphers typically operate on smaller units of plaintext, usually bits. The encryption of any particular plaintext with a block cipher will result in the same ciphertext when the same key is

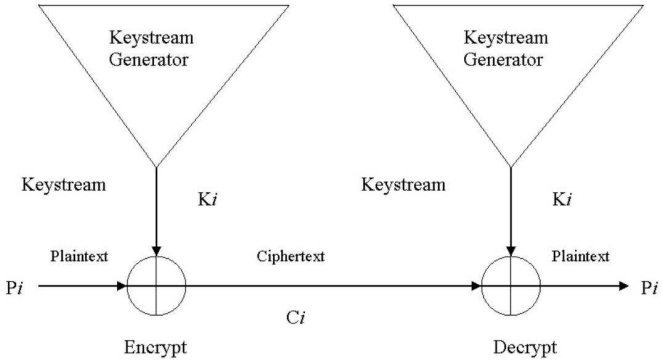

Figure 1: Stream Cipher

used. With a stream cipher, the transformation of these smaller plaintext units will vary, depending on when they are encountered during the encryption process.

A stream cipher generates what is called a keystream (a sequence of bits used as a key). Encryption is accomplished by combining the keystream with the plaintext, usually with the bitwise XOR operation. The generation of the keystream can be independent of the plaintext and ciphertext, yielding what is termed a synchronous stream cipher, or it can depend on the data and its encryption, in which case the stream cipher is said to be self-synchronising [11]. Most stream cipher designs are for synchronous stream ciphers.

The simplest implementation of a stream cipher is shown in Figure 1. A keystream generator outputs a stream of bits: K_1, K_2, K_3 ....., K_i. This keystream is XORed with a stream of plaintext bits, P_1, P_2, P_3, ..., P_4, to produce the stream of ciphertext bits:

$$C_i = P_i \oplus K_i$$

At the decryption end, the ciphertext bits are XORed with an identical keystream to recover the plaintext bits:

$$P_i = C_i \oplus K_i$$

since

$$P_i \oplus K_i \oplus K_i = P_i$$

It is what is inside the generator that defines the level of security given in the keystream. Obviously if there is an endless run of zeros then the ciphertext will equal the plaintext, but if the generator produces a stream of random bits then the security given is the same

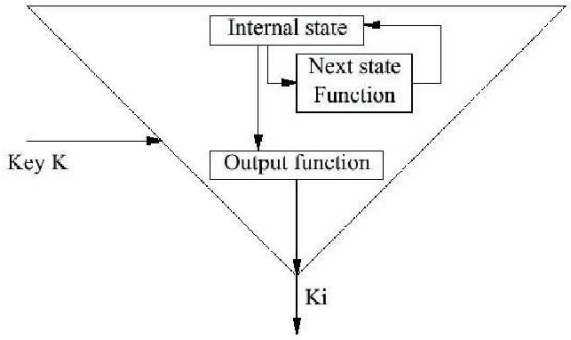

Figure 2: Inside a Keystream Generator

as that of a one-time pad, ie perfect security.

The reality of a stream cipher lies somewhere between the simple XOR and the one-time pad [29]. The keystream generator produces a bit stream that looks random, but is actually a deterministic stream that can be reproduced and decryption time. The idea is that the closer the bit stream is to being random the harder it will be for an attacker to cryptanalyse the stream.

All stream ciphers have keys for added security. The output of the keystream generator is a function of the key. A keystream generator has three basic parts, as illustrated in Figure 2. The internal state describes the current state of the keystream generator. Two keystream generators, with the same key and the same internal state, will produce the same keystream. The output function takes the internal state and generates a keystream bit. The next-state function takes the internal state and generates a new internal state [29].

Examples of stream ciphers algorithms include the Linear Congruential Generator, described in section 2.4, the RC4 algorithm, described in section 2.2 and the A5/1 algorithm. The A5/1 algorithm is the stream cipher algorithm chosen for use in the experiments of this report and is discussed in section 3. Also discussed in section 2.3 is the use of linear feedback shift registers within stream ciphers.

2.1 Output-Feedback Mode and Cipher-Feedback Mode

In OFB mode, the ciphertext does not even go anywhere near the encryptor (see Figure 3). An initialisation vector (IV) is provided; at each step it is encrypted and the output is both XORed with the plaintext and fed back to the encryptor.

OFB has the advantage that the entire output sequence can be calculated before the message is seen, speeding up the encryption process if the sequence can be held se-

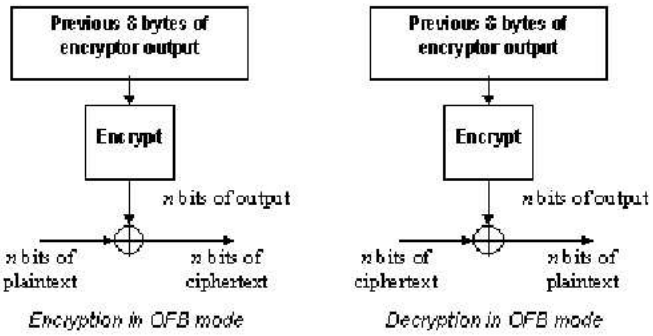

Figure 3: Output-Feedback Mode

curely on disk in the meantime. By analogy with cipher feedback mode (CFB), any number of bits could be fed back from the output into the encryptor; however, the only safe amount is 64 bit (the block size).

A transmission error changing one bit in the ciphertext affects only that bit in the plaintext, making OFB by far the most resistant of the modes to this type of error. A transmission error adding a bit to the ciphertext is irrecoverable. OFB is the mode of choice for noisy transmission environments. However, because it simply XORs the original text, any attacker who knows the plaintext can change the ciphertext to decrypt to any result they want.

Block ciphers can also be implemented as self-synchronising stream ciphers; this is called cipher feedback mode (CFB). In CFB mode data can be encrypted in units smaller than the block size.

In CFB mode, the previous ciphertext block is encrypted, and the output produced is combined with the plaintext block using XOR to produce the current ciphertext block (as shown in Figure 4). Note that the previous eight bytes of ciphertext are always encrypted no matter how many bits of ciphertext are produced at every step.

CFB mode is as secure as the underlying cipher and plaintext patterns are concealed in the ciphertext by the use of the XOR operation. Plaintext cannot be manipulated directly except by the removal of blocks from the beginning or the end of the ciphertext.

With CFB mode and full (64-bit) feedback, when two ciphertext blocks are identical, the outputs from the block cipher operation at the next step are also identical. This allows information about plaintext blocks to leak. When using full feedback, the speed of encryption is identical to that of the block cipher, but the encryption process cannot be easily parallelled.

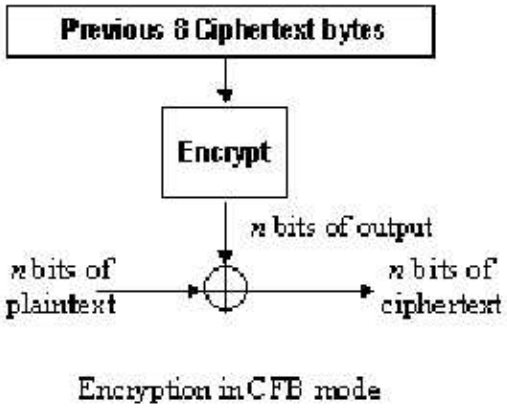

Figure 4: Cipher-Feedback Mode

CFB mode is most frequently used with a feedback size of eight bits, encrypting one byte at a time. Under these circumstances it runs at an eighth of the speed of the parent block cipher.

2.2 RC4

A commonly used stream cipher is RC4. Interestingly, certain modes of operation of a block cipher effectively transform it into a keystream generator and in this way, any block cipher can be used as a stream cipher; as in DES in CFB or OFB modes. However, stream ciphers with a dedicated design, for example the A5 algorithm, are typically much faster [30].

RC4 is a variable-key-size stream cipher developed in 1987. In 1994 the algorithm details were leaked into cyberspace and confirmed as correct by readers who had legal copies of the algorithm.

The following description of the algorithm is taken from Bruce Schneier's book, Applied Cryptography [29]. The algorithm works in Output-Feedback Mode. The keystream is independent of the plaintext. It has $8*8$ substitution box (S-box): $S_0, S_1, ..., S_{255}$. the entries are a permutation of the numbers 0 through 255, and the permutation is a function of the variable-length key. It has two counters, i and j, initialised to zero. For the random byte generation, the following is done:

$i = (i+1) \bmod 256$
$j = (j+S_i) \bmod 256$
swap S_i and S_j

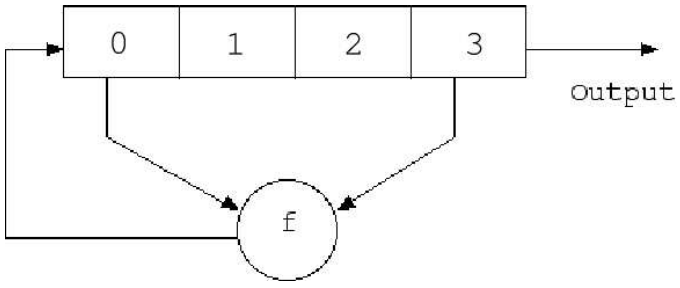

Figure 5: Linear Feedback Shift Register

$t = (S_i + S_j) \bmod 256$
$K = S_t$

The byte K is XORed with the plaintext to produce ciphertext or XORed with the ciphertext to produce plaintext. The encryption is fast, about ten times faster than DES. The swap operation makes recovery of table S very difficult.

2.3 Linear Feedback Shift Registers

One of the two main parts of a linear feedback shift register (LFSR) is the shift register (the other being the feedback function). A shift register is a device whose identifying function is to shift its contents into adjacent positions within the register or, in the case of the position on the end, out of the register. The position on the other end is left empty unless some new content is shifted into the register. Feedback Shift Registers are a commonly used method of producing pseudo-random sequences and are by themselves stream cipher generators [10].

The output of the shift register is one bit, usually the least significant bit. The period of a shift register is the length of the output sequence before it starts repeating.

Stream ciphers made up of LFSRs have been favoured by cryptographers as they are easily implemented in digital hardware [7]. The simplest kind of feedback shift register is the LFSR. The feedback function is the XOR of certain bits in the register. The list of these bits is called the tap sequence. Figure 5 shows a four bit LFSR tapped at the first and fourth bit. The bits at bit one and four are XORed and the result is fed back into the register when the register shifts one bit to the right, in this case. Bit four is the output when the register shifts.

An n-bit LFSR [1] can be in one of $2^n - 1$ internal states. This means that it can in theory,

[1] The length of a shift register is worked out in bits, if it is n bits long, it is called an n-bit shift register

generate a $2^n - 1$ bit long pseudo-random sequence before repeating. Only LFSRs with certain tap sequences will cycle through all $2^n - 1$ internal states and these are called maximal-period LFSRs [29].

For it to be a maximal-period LFSR the polynomial formed from a tap sequence plus the constant one must be a primitive polynomial mod 2. For this report the mathematical theory behind this will not be analysed. The polynomial associated with the LFSR in figure 5 is:

$X^4 + X^3 + 1$

and it is a primitive polynomial. This means that because the first and fourth bits of this four bit register are XORed together to generate the new bit the LFSR will be maximal length, that is it will cycle through $2^4 - 1$ values before repeating.

2.4 Linear Congruential Generators

Donald Knuth mentions in his book "The Art of Computer Programming" [13] that the most popular random number generators used are special cases of a scheme introduced by D. L. Lehmer in 1949 called the linear congruential generator (LCG). LCGs are psuedo-random sequence generators of the form:

$X_n = (AX_{n-1} + B) mod M$

in which X_n is the nth number of the sequence, and X_{n-1} is the previous number of the sequence. the variables A, B, and M are constants. The key, or seed, is the value of X_0.

This generator has a period no greater than M. The generator will be a maximal period generator if B and M are relatively prime and the right values of M are chosen. The number generation process is a little faster when $C = 0$, however the restriction of $C = 0$ cuts down on the length of the period of the sequence.

Linear congruential generators cannot be used for cryptography; they are predictable and have been broken by several cryptanalysists [29]. However LCGs remain useful for noncryptographic applications such as simulations. They are efficient and show good statistical behaviour with respect to most reasonable empirical tests.

3 Introduction to GSM and GSM Security

The use of analog cellular telephony grew rapidly, particularly in Europe, during the 1980s. each country developed its own system without regard to compatibility, an undesirable situation in a unified Europe and a problem because the use of different equipment meant that economies of scale could not be realised. In 1982 the Conference of European Posts and Telegraphs (CEPT) formed a study group called the Groupe

Special Mobile (GSM) to study and develop a pan-European public land mobile system.

In 1989, GSM responsibility was transferred to the European Telecommunication Standards Institute (ETSI), and phase I of the GSM specifications was published in 1990. Commercial service was started in mid-1991, and by 1993 there were 36 GSM networks in 22 countries, with 25 additional countries having already selected or considering GSM. By this time GSM was worldwide, and the acronym GSM became redefined as the Global System for Mobile telecommunications.

There were some widely known problems with GSM's analog counter parts. these problems included the possibility of phone fraud through cloning phones and thus calling at someone else's expense, and the possibility of someone intercepting the phone call over the air and eves dropping on the discussion. The GSM system was supposed to correct these problems by implementing strong authentication between the Mobile Station (MS - mobile phone) and the Mobile services Switching Center (MSC) [2], as well as implementing strong data encryption for the over-the-air transmission channel between the MS and the Base Transceiver Station (BTS) [3].

The GSM specifications were designed by GSM in secrecy and were distributed only on a need-to-know basis to hardware and software manufacturers and to GSM network operators. The specifications were never exposed to the public, thus preventing the open science community around the world from studying the enclosed authentication and enciphering algorithms as well as the whole GSM security model.

GSM relied on security by obscurity, that is the algorithms would be harder to crack if they were not publicly available. According to the open scientific community, one of the basic requirements for secure cryptographic algorithms is that the security of the crypto system lies solely on the key. this is known as Kerckhoffs' assumption after Auguste Kerckhoffs publicly voiced his opinion in 1883 on what is now known as this security by obscurity.

Kerckhoffs was the first to publicly identify the weakness of security by obscurity. The admonition not to rely on obscurity in security systems is often called "Kerckhoffs' Principle" or "Kerckhoffs' Law". He advocated what is now called "open systems": instead of relying on the obscurity of the cryptosystem's underlying operation, Kerckhoffs argued that designers should assume that the attacker will either know or be able to deduce how the system works. Instead of relying on obscurity, he argued, security should depend on the strength of keys. In the event of a breach, only the keying material would need to be replaced, not the whole system [32].

The algorithm in question should be made publicly available, so that the algorithm

[2]The MSC performs the switching functions of the network. It also provides a connection to other networks

[3]The BTS is a base station the MS communicates with

is exposed to the scrutiny of the public. According to the general opinion no single entity can employ enough experts to compete with the open scientific community in cryptanalysing an algorithm [21]. Following on form this the algorithms designed and implemented in secrecy will probably be cryptographically weak and contain design faults. Eventually the GSM algorithms leaked out and have been studied extensively ever since by the open science community.

3.1 Description of GSM Security Features

Security in GSM consists of the following aspects: subscriber identity authentication, subscriber identity confidentiality, signalling data confidentiality, and user data confidentiality. The subscriber is uniquely identified by the International Mobile Subscriber Identity (IMSI). This information, along with the individual subscriber authentication key (Ki), constitutes sensitive identification credentials analogous to the Electronic Serial Number (ESN) in analog systems such as AMPS and TACS. The design of the GSM authentication and encryption schemes is such that this sensitive information is never transmitted over the radio channel. Rather, a challenge-response mechanism is used to perform authentication. The actual conversations are encrypted using a temporary, randomly generated ciphering key (Kc). The MS identifies itself by means of the Temporary Mobile Subscriber Identity (TMSI), which is issued by the network and may be changed periodically (i.e. during hand-offs) for additional security.

The security mechanisms of GSM are implemented in three different system elements; the Subscriber Identity Module (SIM), the GSM handset or MS, and the GSM network. The SIM contains the IMSI, the individual subscriber authentication key (Ki), the ciphering key generating algorithm (A8), the authentication algorithm (A3), as well as a Personal Identification Number (PIN). The GSM handset contains the ciphering algorithm (A5). The encryption algorithms (A3, A5, A8) are present in the GSM network as well. The Authentication Center (AUC), part of the Operation and Maintenance Subsystem (OMS) of the GSM network, consists of a database of identification and authentication information for subscribers. This information consists of the IMSI, the TMSI, the Location Area Identity (LAI), and the individual subscriber authentication key (Ki) for each user. In order for the authentication and security mechanisms to function, all three elements (SIM, handset, and GSM network) are required. This distribution of security credentials and encryption algorithms provides an additional measure of security both in ensuring the privacy of cellular telephone conversations and in the prevention of cellular telephone fraud.

Figure 6 demonstrates the distribution of security information among the three system elements, the SIM, the MS, and the GSM network. Within the GSM network, the security information is further distributed among the authentication center (AUC), the home location register (HLR) and the visitor location register (VLR). The AUC is responsible for generating the sets of RAND, SRES, and Kc which are stored in the HLR and VLR for subsequent use in the authentication and encryption processes.

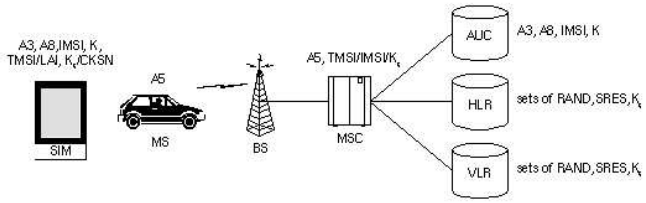

Figure 6: Distribution of Security Features in the GSM Network

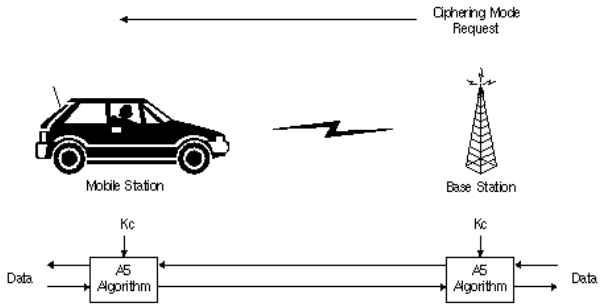

Figure 7: Using the A5

Figure 7 shows how encrypted voice and data communications (over-the-air traffic) between the MS and the network is accomplished through use of the ciphering algorithm A5. Encrypted communication is initiated by a ciphering mode request command from the GSM network. Upon receipt of this command, the mobile station begins encryption and decryption of data using the ciphering algorithm (A5) and the ciphering key (Kc). The ciphering key is produced by the A8 algorithm [16].

Figure 8 shows the encryption process. Each frame in the over-the-air traffic is encrypted with a different keystream. This keystream is generated with the A5 algorithm. The A5 algorithm is initialised with the Kc and the number of the frame to be encrypted, thus generating a different keystream for every frame. This means that one call can be decrypted when the attacker knows the Kc and the frame numbers. The frame numbers are generated implicitly, which means that anybody can find out the frame number at hand. The same Kc is used as long as the MSC does not authenticate the MS again, in which case a new Kc is generated. In practice, the same Kc may be in use for days. The MS authentication is an optional procedure in the beginning of a call, but it is usually not performed. Thus, the Kc is not changed during calls.

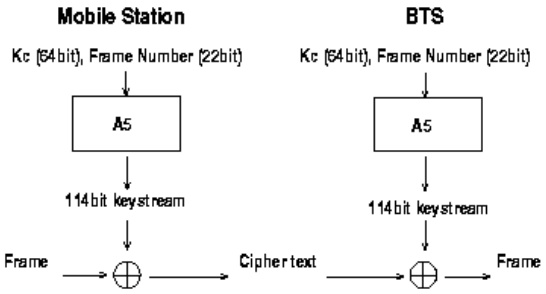

Figure 8: Encrypted Voice and Data communications Using the A5 Algorithm

3.2 The A5/1 Algorithm

The A5 algorithm has two main variants the A5/1 and the A5/2. The A5/1 is the stronger of the two and its design was leaked in 1994. Most of the details of the A5/1 encryption algorithm are known as a British telephone company gave all of the documentation to Bradford University without signing a non-disclosure agreement. This information eventually leaked out and found its way to the internet [29]. The exact design of both the A5/1 and the A5/2 was reverse engineered from an actual GSM telephone. This design was confirmed by the GSM organisation [1].

The A5 consists of three LFSR registers of varying lengths, 19, 22 and 23. All of the feedback polynomials are sparse [6]. The output of the algorithm is the XOR of all three registers taking into account the clock control that the algorithm uses.

Figure 9 shows the three short LFSRs used in the A5/1, they are labelled R1, R2 and R3. The rightmost bit in each register is labelled as bit zero. The taps of R1 are at bit positions 13,16,17 and 18. The taps of R2 are at bit positions 20 and 21 and the taps of R3 are at bit positions 7, 20, 21 and 22.

These three registers are maximal length (or maximal period) linear feedback shift registers with periods $2^{19}-1$, $2^{22}-1$, and $2^{23}-1$, respectively [1]. They are clocked in a stop/go fashion using a majority rule. The description of this majority rule is that each register has a single tap that is used in the clocking process. These taps are bit 8 for R1, bit 10 for R2 and bit 10 for R3 and are shown as C1, C2 and C3 in Figure 9. As explained previously the registers are shifted if the bit at the tap of that register is part of the majority. So for example a register with the tap bit set at 0 will shift if all of the tap bits are 0 or two out of three of the tap bits are 0. At each step either two or three registers will be clocked. So if C1 = 0, C2 = 0, and C3 = 1, the majority would be 0 and registers R1 and R2 would clock, that is they would shift whereas register R3 would not.

The A5/2 is similar to A5/1 but has a fourth LFSR R4 of length 17 bits with taps at 17

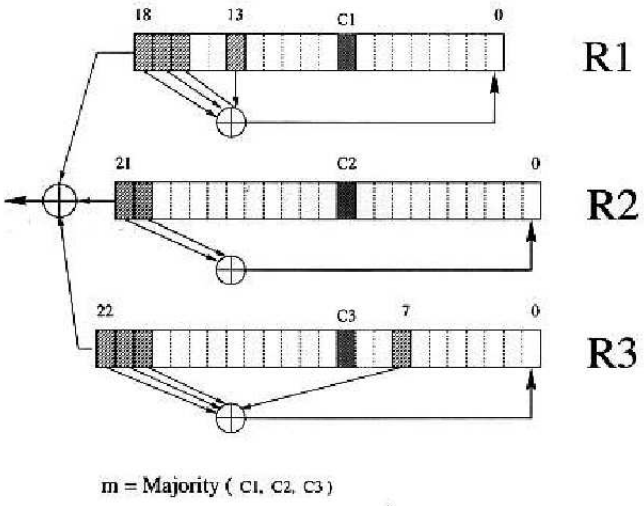

Figure 9: The A5/1 Stream Cipher

and 16 which controls the clocking in place of the majority function using bits 10,3, and 7 for R1, R2, and R3. R4 is always clocked.

There is a great deal of controversy over the use of A5, with claims of government interference in order to ensure a sufficiently weak algorithm. This seems to have been confirmed by the publication of various known attacks including:

- A proposed attack based on guessing the 41 bits in the shorter R1 and R2 registers, and deriving the 23 bits of the longer R3 register from the output [1].
- It has been found that the A5/1 is vulnerable to an attack which requires a working factor approximating 2^{40} [9].

The A5 has also been said to be very efficient and that its weaknesses include the fact that its registers are short enough to make an exhaustive search feasible. Longer shift registers and denser feedback polynomials should make this algorithm more secure. However tests have been done on the algorithm to show that its keystreams are sufficient to pass the most standard statistical tests for randomness [29]. Nevertheless, and whether through deliberate interference or not, A5/1 is vulnerable to attack and so can be considered as a relatively robust stream cipher to which improvements can be made.

It is therefore a suitable “live” keystream generator with which to test the claim made by the authors of MSK, that is to assess whether the addition of MSK exponentially improves the randomness of a keystream generator.

4 Multiple Excess-S Modulo-K (MSK) coding scheme

The Multiple Excess-S Modulo-K (MSK) code has been proposed by M.L Keshariya and S. Keshariya [12] as a scheme for improving the randomness of a stream cipher by using their developed MSK coding scheme. In fact the authors suggest that this scheme can exponentially enhance the security of stream ciphers.

4.1 Excess-3 Code

The Excess-3 code is a coding scheme used with Binary Coded Decimal (BCD) numbers, in which the value 3 is added to each input to produce the coded output. Input BCD digits 0-9 are transformed into the values 3-12 that is:

$y = x + 3$

Excess-3 had been used in the Bell Telephone Laboratories Model I relay calculator built in 1940 and was subsequently adopted by the first Univac-1 computer [3]. Excess-3 was chosen for the UNIVAC because it simplified some aspects of digit-by-digit decimal operations - for example, the ones complement of an Excess-3 number is the nines complement of that number in decimal and made the carries come out right for digit-by-digit decimal addition.

4.2 Excess-S Code

The authors of the MSK algorithm present a generalisation of the Excess-3 code and call it Excess-S code. It can be derived from BCD by adding an equivalent binary expression of S, to a decimal number. Unlike the Excess-3 coding scheme, Excess-S encoding may require an increase in symbol bits.

Mathematically Excess-S code = BCD code + $(S)_2$

or $y = (x)_2 + (S)_2$

If x is a decimal digit then y can be calculated by the above equation. It will have n bits depending on the values of S and x.

The Excess-S scheme is then extended by the authors to the idea of an Excess-S code over Modulo K. If x is a decimal digit the concept of Excess-S code over a modulo K means that every digit will be represented by a uniform n-bit representation. The authors therefore define Excess-S code as an n-bit code depending on the value of x and K as follows:

$y = x + S(modK)$

So y is an n-bit Excess-S code over mod K for corresponding value of x, that is y is constrained into a certain number of n-bits limiting it to a set of integers 0-K.

The authors show in their paper that this scheme can be used to describe the Caesar cipher by assigning the values K=26 and S=3. The authors also suggest that this scheme can be used to describe polyalphabetic ciphers such as Vigniere, Beafort and Porta.

4.3 MSK

Finally, the Multiple Excess Mod (K) scheme is defined as:

$y = Mx + S(modK)$

By constraining the values of M and K to be such that gcd(M,K)=1, that is M and K are relatively prime, every value of x will result in a unique value of y and so the scheme can be used to encode/decode information.

The Euclidean algorithm is used in the implementation of MSK to determine this greatest common divisor between M and K (i.e the largest integer that divides both numbers). Two numbers are called coprime or relatively prime if their greatest common divisor equals 1. For example, 9 and 28 are relatively prime.

First x is to be converted into multiple code (Mx) and then transformed into Excess-S code over Modulo K as the operations are not commutative that is: $Mx + S(modK) <> M(x+S)(modK)$

This process is then applied to the generated keystream, for example from the generated keystream produced by the A5/1 algorithm. The keystream is to be considered as a series of n bit values (ie x) where $2^n = K$.

The MSK coding scheme looks very familiar to the linear congruential generator (see section 2.4) however the MSK scheme is not proposed as a generator but as an encoding scheme. Consequently the requirement in applying MSK to an input stream is that M must be relatively prime to K, rather than S being relatively prime to K. With the linear congruential generator one of the requirements for the generator to have a maximal period is that S must be relatively prime to K.

The following is an example of why M must be relatively prime to K:

(1) M0 mod K = 0 (ie when x=0) if we can find a value such that Mx mod K = 0 for x $>$ 0, then there is a problem

(2) For any value S, M0 +S mod K = S (ie when x=0) if a value can be found such that Mx+S mod K = S for x$>$0, then there is the same problem problem

(3) Let gcd(M,K) be n. It transpires that MK/n = 0 mod K for all n$>$1 and so x=K/n will always cause a problem.

The following provides proof that M must be relatively prime to K:

- Consider y = xM mod K where M<K
- Let n = gcd(M,K)
- Then for some nonzero m,k M = nm and K = nk, ie k = K/n and m = M/n
- Consider the value K/n which is < K. Then the original equation becomes y = K/n * M mod K
- But K/n * M is nk/n * mn = nkmn/n = nkm = Km, so y = Km mod K = 0
- But x=0 also returns 0, so there are duplicate values
- In the special case of n=1, ie M,K relatively prime, then the value x=0 and x=K are, mod K, the same so there are no problems
- Therefore to have a substitution table gcd(M,K) must = 1

4.4 Generation of MSK Codes

As an example all MSK codes for values of x ranging from 00 to 63 have been calculated. As shown in Table 1 MSK code represented by *y* as output in six bits (U,V,W,X,Y,Z) from initial input values of *x* in six bits (A,B,C,D,E,F). Taking values that are suggested in the authors' paper, M=21, S=47, K=64 then:

$$y = 21.x + 47 (mod 64)$$

INPUT							OUTPUT						
x	A	B	C	D	E	F	y	U	V	W	X	Y	Z
00	0	0	0	0	0	0	47	1	0	1	1	1	1
01	0	0	0	0	0	1	04	0	0	0	1	0	0
02	0	0	0	0	0	1	25	0	0	0	1	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
62	1	1	1	1	1	0	05	0	0	0	1	0	1
63	1	1	1	1	1	1	26	0	1	1	0	1	0

Table 1: Generation of MSK code for M=21, S=47 and K=64

4.5 MSK as an Enhancement to Stream Ciphers

As shown in Figure 10 the authors of the MSK coding scheme propose that MSK codes can be used with a stream cipher in one of four methods:

A By applying MSK to the keystream prior to its XOR with the plaintext

B By applying MSK to the plaintext prior to its XOR with the keystream

C By applying two different MSK transformations to the keystream and the plaintext

D By applying two different MSK codes and introducing a non-linear combining function in place of the traditional XOR operation

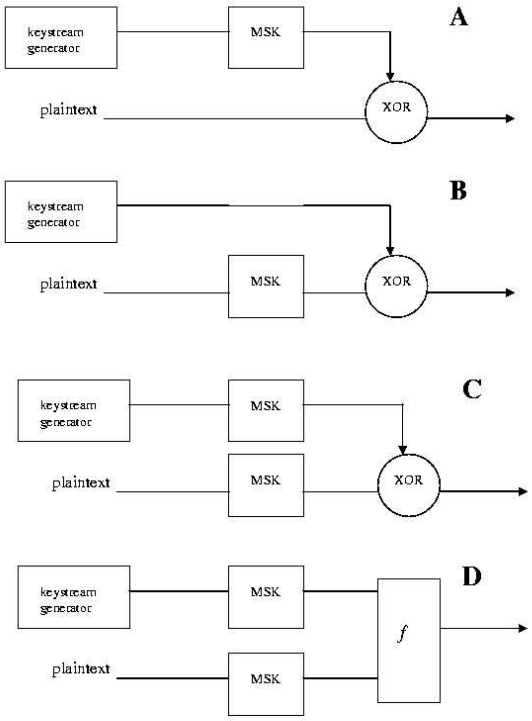

Figure 10: Four Methods of Applying MSK

The author's have made the claim that the MSK coding scheme can exponentially enhance the security of stream ciphers. To test this claim method A will be used where tests will be done on the keystream before and after the MSK coding scheme has been used. As illustrated in Figure 11 the A5/1 algorithm will be used as the keystream generator. The keystream produced by the A5/1 algorithm will be compared with the new keystream produced by the MSK enhance A5/1 keystream.

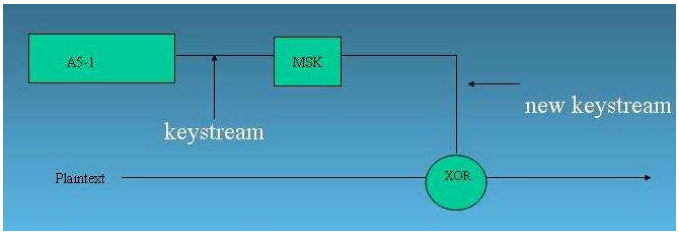

Figure 11: Applying MSK to the A5/1 Algorithm

5 Randomness and Psuedorandomness

All elements of a random bit sequence are generated independently of each other, and the value of the next element in the sequence cannot be predicted, regardless of how many elements have already been produced [18].

Random and pseudorandom numbers are needed for many cryptographic applications. For example, common cryptosystems employ keys that must be generated in a random fashion. Many cryptographic protocols also require random or psuedorandom inputs at various points, for example for auxiliary quantities used in generating digital signatures or for generating challenges in authentication protocols.

All elements of a random bit sequence are generated independently of each other, and the value of the next element in the sequence cannot be predicted, regardless of how many elements have already been produced [18].

There are two basic types of generators used to produce random sequences: random number generators and psuedorandom number generators. A random number generator uses a non-deterministic source (i.e some unpredictable physical source) to produce random bits. A psuedorandom number generator produces a sequence of bits from an initial value called a seed using a known algorithm. For cryptographic applications, both of these generator types produce a stream of zeros and ones that may be divided into substreams or blocks of random numbers [4].

5.1 Random Number Generators

It has been said that it is impossible to produce something truly random on a computer. Donald Knuth quotes John von Neumann as saying: "*Anyone who considers arithmetic methods of producing random digits is, of course, in a state of sin*" [13]. Another author has said "*Computers are deterministic beats: Stuff goes in one end, completely predictable operations occur inside, and different stuff comes out the other end. Put the same stuff in on two separate occasions and the same stuff comes out both times. Put the same stuff into two identical computers, and the same stuff comes out of both of them*" [29].

A computer can only be in a finite number of states, and this "stuff" that comes out will always be a deterministic function of the "stuff" that went in and the computer's current state. That means that any random-number generator on a computer is, by definition, periodic. Periodic means that eventually the output sequence will repeat. Anything that is periodic is predictable. If something is predictable, it can't be random. "*A true random-number generator requires some random input; a computer can't provide that*" [29].

For cryptographic purposes, the output of RNGs needs to be unpredictable. However some physical sources (eg date/time vectors) are quite predictable. These problems may be minimised by combining outputs from different types of sources to use as the inputs for an RNG. However, the resulting outputs from an RNG may still be deficient when evaluated by statistical tests. Also the production of high quality random numbers may be too time consuming, which is not good when a large number of random numbers are needed, so instead psuedorandom number generators may be preferable [18].

5.2 Psuedo-Random Sequences

The best a computer can produce is a pseudo-random generator. The sequence produced by this generator looks random and the period should be long enough so that a finite sequence of reasonable length, ie the sequence that is actually used, is not periodic. The idea is that if a billion random bits are needed don't choose a sequence with a periodic length of sixteen thousand bits. These relatively short non periodic sequences should be as close as possible to a random sequence, so close that it is hard to tell the difference. For example they should have about the same number of ones and zeros, and they should not be compressible this is discussed further in the statistical tests section 6. There are many discussions of generators in literature, including many on the linear congruential generator as discussed in section 2.4.

Non-random strings are the strings that possess some kind of regularity. Regularity is a good basis for compression. Therefore randomness means the absence of any compression possibility; it corresponds to maximum information content (because after dropping any part of the string, there remains no possibility of recovering it) [5].

Cryptographic applications demand much more of a pseudo-random-sequence generator than do most other applications. Cryptographic randomness doesn't mean just statistical randomness, although that is part of it. For a sequence to be cryptographically secure pseudo-random, it must be unpredictable. It must be computationally infeasible to predict what the next random bit will be, given complete knowledge of the algorithm or hardware generating the sequence and all of the previous bits in the stream [29].

Cryptographically secure psuedo-random sequences should not be compressible, unless the key is known. The key is generally the seed used to set the initial state of the generator. Like any cryptographic algorithm, cryptographically secure pseudo-

random-sequence generators are subject to attack. Just as it is possible to break an encryption algorithm, it is possible to break a cryptographically secure pseudorandom generator. Cryptography involves making these generators resistant to such attacks.

5.3 Unpredictability

Random and pseudorandom numbers generated for cryptographic applications should be unpredictable. In the case of PRNGs, if the seed is unknown, the next output number in the sequence should be unpredictable in spite of any knowledge of previous random numbers in the sequence. This is known as forward unpredictability. It should also not be feasible to determine the seed from knowledge of any generated values (i.e backward unpredictability is also required). No correlation between a seed and any value generated from that seed should be evident; each element of the sequence should appear to be the outcome of an independent random event whose probability is 1/2 [18].

6 Statistical Tests for Randomness

A means of objectively comparing the quality of randomness produced by keystream generators is required in order to assess the effect of the MSK coding on the quality of a stream cipher. The major testing schemes for assessing randomness are reviewed and an appropriate test regime selected.

6.1 The NIST SP800-22 Test Suite

The National Institute of Standards and Technology (NIST) has published Federal Information Processing Standard 140-2 which defines four tests to be passed by any approved cryptographic (psuedo-)random number generator. A more recent document, NIST's Special Publication 800-22 [18] describes a wider test suite of sixteen statistical tests recommended for use by the US Federal Government Agencies in testing cryptographic keystreams.

These tests are designed to provide assurance of the quality of randomness in binary sequences that are produced by random or pseudo-random number generators intended for use in cryptography. The tests also determine the uniformity, scalability, and consistency of results across a large sample of data and (in the case of pseudo-random number generators) seeds.

The tests provide an indication of the existence or absence of a pattern of some form in the sample data. They are all independent, each testing for a different kind of pattern and any one or combination of tests can be run. The tests are designed to all return a common p-value indicating the acceptable probability of failure of the test. The standard value used in the SP800-22 tests is 0.01, ie less than 1 in 100 samples should fail the test (a 99% confidence level).

This p-value can be interpreted as the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested. So a p-value of 1 means there is a 100% probability, that is it is always true, that the tested sequence is more random than that generated by a perfect random number generator. A p-value of 0 indicates that there is no chance that a perfect random number generator would have produced an inferior sequence.

Statistical tests are commonly used to detect deviations of a binary sequence from randomness. However, some degree of failure is expected even in a good random number generator. Further, as previously discussed, there are two types of random numbers, those produced by a true random number generator and those produced by a pseudo-random number generator. A true random number generator uses some form of non-deterministic source of bits, and these will typically be processed through a distillation process to produce random numbers. The distillation process is needed to overcome any entropy weakness in the source. Pseudo-random numbers are generated from initial (seed) values which are processed in some form of deterministic manner. It is not uncommon for a pseudo-random generator to provide better statistical results for randomness than a true random source. Consequently, the results of running any statistical tests on a sample of random numbers need to be interpreted with the above comments in mind.

These tests can be used as a first step in determining whether or not a generator is suitable for a particular cryptographic application. It should be noted that these statistical tests do not serve as a substitution for cryptanalysis, as no set of these tests can absolutely certify a generator as appropriate for usage in a particular application; nevertheless, these tests do serve to provide a high level of confidence in the absence of cryptanalysis. The use of cryptanalysis to assess the quality of a random or pseudo-random number generator is outside the scope of this paper.

The following is a brief description of the tests used in the NIST test suite. For further details refer to Section 3 of the NIST SP 800-22 document.

Frequency (Monobit) Test The proportion of zeros and ones across the binary sequence is measured in this test with the purpose of determining whether the number of ones and zeros is that of what would be expected in a true random sequence. The number of ones and zeros in a sequence should be approximately the same. A successful test shows that there is no evidence to suggest that the sequence is non-random. All subsequent tests depend on the passing of this test.

Frequency Test within a Block This test determines the proportion of ones within M-bit blocks. Under the assumption of randomness the number of ones within a block should be approximately M/2. For a block size of one bit (M=1) the test becomes the Frequency Test (see above).

Runs Test A run is an uninterrupted sequence of identical bits. The test looks at the total number of runs in a sequence. The test determines whether the number of runs of zeros or ones of various lengths is that of which is expected for a random

sequence. In particular the tests looks at whether the oscillation between the runs of zeros and ones is to fast or too slow.

Test for the Longest Run of Ones in a Block This test measures the longest run of ones within M-bit blocks. The purpose of the test is to determine whether the length of the longest run of ones is that of would be expected for a random sequence. The length of ones only needs to be tested because an irregularity in the length of ones would also imply an irregularity in the length of the longest run of zeroes.

Binary Matrix Rank Test This test checks for linear dependencies among fixed length substrings of the original binary sequence.

Discrete Fourier Transform Test In this test peak heights in the Discrete Fourier Transform of the sequence are measured. The purpose is to detect periodic features, that is, repetitive patterns that are near each other, in the tested sequence that would indicate a deviation from the assumption of randomness.

Non Overlapping Template Matching Test In this test pre-specified target string occurrences are measured in the sequence. This test detects generators that produce too many occurrences of a given non-periodic pattern. A window is used of a certain bit size, when a pattern is found the window is reset to the bit after the found pattern otherwise the window slides one bit along in the sequence.

Overlapping Template Matching Test This test is the same as the one above except that when a pattern is found the window moves one bit along in the sequence.

Maurer's "Universal Statistical" Test This test looks at the number of bits between matching patterns with the purpose of detecting whether or not the sequence can be significantly compressed without loss of information. The sequence is considered to be non-random if the sequence can be significantly compressed.

Lempel-Ziv Compression Test This test determines how far the tested sequence can be compressed. The sequence is considered to be non-random if it can be significantly compressed.

Linear Complexity Test This test focuses on the length of a linear feedback shift register (LFSR). An LFSR that is too short implies non-randomness as random sequences are characterised by longer LFSRs. The test determines whether the sequence is complex enough to be considered random.

Serial Test If a sequence is random every m-bit pattern will have the same chance of appearing as every other m-bit pattern. This test determines whether the number of overlapping patterns is approximately the same as would be expected for a random sequence.

Approximate Entropy Test This test measures the frequency of all possible overlapping m-bit patterns across the sequence as the Linear Complexity Test (above) does. This test compares the frequency of overlapping blocks of two adjacent lengths (m and m+1) against the expected result for random sequence.

Cumulative Sums Test This test determines whether the cumulative sum of partial sequences occurring in the test sequence are either too large or too small according to the expected results for random sequence cumulative sums.

Random Excursions Test This test determines the number of cycles. ie the sequences of cumulative values starting at and returning to zero, having a specific number of visits to each state.

Random Excursions Variant Test This test determines the total number of times a particular state is visited in a cumulative walk through the whole sequence.

7 Current Literature on Statistical Tests

There are many statistical tests described in literature for checking whether a sample is random or not. A number of the papers on the subject of quality of randomness use or suggest the use of the null hypothesis to determine whether a pseudorandom number generator is acceptably close to a perfect RNG [14], as does the NIST Statistical test suite through its use of threshold p-values to accept or reject a sample at a particular confidence level.

Maurer provides a universal test for randomness which uses the measure of compressibility of a stream of random numbers. He suggests that this test parameter is closely related to the random number generator's per-bit entropy and is the correct quality measure for cryptographic applications.

Maurer's test has often been chosen from many statistical tests for comparing pseudorandom generated streams because the test is universal, meaning that it detects arrange of defects found by various other single tests [22]. The main idea behind Maurer's universal test is that it should not be possible to significantly compress (without loss of information) the output sequence of a random bit generator. Therefore, if a sample output sequence of a bit generator can be significantly compressed, the generator should be rejected as being defective and significantly non-random. Instead of actually compressing the sequence, the universal statistical test computes a quantity that is related to the length of the compressed sequence.

This test can reasonably be argued to comprise most defects that can realistically be expected in a practical implementation of a random bit generator. Another feature of the universal test is that it measures the actual cryptographic significance of a possible defect, namely the per-bit redundancy [17].

Goldreich [8] also states that "*typically a good way to quantify the amount of randomness, or unpredictability in a distribution, is to compute its entropy*". Other authors have acknowledged that entropy is a measure of possible patterns present within random data and is therefore an obvious test to use for testing the randomness of pseudorandom generated streams [31].

Randomness can be assessed via the Approximate Entropy test (introduced in section 6.1). This test will provide a computable measure of sequential irregularity, applicable to single sequences of both (even very short) finite and infinite length and is a productive way to evaluate cryptosystems [28].

Other authors have commented that the entropy of a random number generator is an easy to calculate and significant measure for comparison of pseudorandom generated streams and for applicability of pseudorandom generators [26]. It has also been acknowledged that the approximate entropy test can be applied to the problem of testing for randomness a string of binary bits. To measure the degree of randomness of observed sequences it has been suggested to use a general characteristic, that is the Approximate Entropy [23]. This concept has been introduced by S. Pincus and co-authors [27].

The importance of using the Approximate Entropy test is strengthened also by the fact that the procedures based on randomness test via approximate entropy form now a part of the battery of empirical tests for randomness developed by NIST (refer section 6.1). The Approximate Entropy test has played an important part in the investigation of various existing random number generators, such as data Encryption Algorithm, Secure Hash Algorithm, Digital Signature Algorithm and Blum, Blum and Shub generator [23].

In his advanced literature study, Richter [22] specifically addresses the problem of comparing stream ciphers, and suggests the use of the MonoBit Test and Maurer's Universal test as being adequate. The mononbit frequency test (also called the frequency test) is the simplest and most fundamental of the randomness tests. It measures the relative amount of 1's and 0's appearing in the keystream, since a truly random source exhibits equal 1's and 0's as the keystream grows to infinity. As mentioned in section 6.1 all subsequent tests used will depend on the passing of this test.

8 Choosing the Metric

There are a number of commonly accepted tests to determine whether a sample should be considered random or not, The Frequency Test and Maurer's Universal Test appearing most commonly in literature. In addition, there are two metrics for assessing the quality of randomness that appear in the literature, the entropy and the complexity of the sample [25]. As mentioned in section 7 entropy has been described as the correct measure by multiple authors.

For this report the Frequency Test was selected as an initial null hypothesis filter to confirm that the sample is acceptable. Having past this test, Maurer's Universal Test has been selected as a secondary null hypothesis test to reject any sample which displays any of the universal defects in randomness. The NIST statistical test suite calculations of the p-value are used for rejecting the null hypothesis with a threshold of 0.01 in both

the Frequency Test and Maurer's Universal Test.

Having then accepted that the sample is an acceptable pseudorandom sample, the Approximate Entropy Test is used to assess the quality of the random stream based on its entropy. These three tests were obtained from the NIST statistical test suite. The chi squared statistic, that is used to obtain the respective p-value in the Approximate Entropy Test, is used for the comparison of the A5/1 and X-generation (see section 9.1) produced keystreams with the MSK enhanced keystreams. The better the randomness of the sample, the closer this metric approaches the ideal value of $ln(2)$ or 0.693147 [24].

Given these filters and metrics, the extent to which the MSK coding scheme provides improvements in randomness can be assessed.

9 The Experiments

9.1 The X-Generation

For experimental purposes a set of four simple 32-bit LFSR-based stream generators were developed. These generators were used to create a set of weak random samples with which to carry out an initial assessment of MSK's ability to enhance poor randomness.

These simple generators (named the X-Generation for this report) were developed with arbitrarily selected three, four and four taps. The tap positions are shown in the following polynomials:

X1: $X^5 + X^2 + X + 1$

X2: $X^{22} + X^{17} + X^5 + X^2 + 1$

X3: $X^{15} + X^{14} + X^{13} + X + 1$

9.2 Applying MSK to the X-Generation

One Mega bit files where generated from each of the three simple LFSR registers. The sequence within the file for each register was then tested using the selected empirical tests. MSK was then applied to each these sequences and the tests run again on the new sequences.

Tables 2 and 3 show the results of the tests performed on the X1 X-Generation register. Trial numbers 7 and 8 in Table 2 show some anomalies in otherwise consistent results.

The high number of fails and the low entropy values is indicative of a short feedback shift register with sparse non-primitive polynomials. The low values in trial numbers 7 and 8 show that the values are repeating within a very short loop and that the maximal

period is very short. Analysing the file dump of the generated file showed that the period was a two bit cycle in the case of trial 7.

The results show that by applying MSK to the X1 generated stream there are no improvements, in some cases it is possible that the strength of the stream is even reduced. The entropy figures are very low relative to the ideal figure which is 0.693147 (see section 8). This is again indicative of the type of LFSR used and the positioning of the taps. Only in the two cases where the initial stream had very low values, Trials 7 and 8, did the application of MSK improve the entropy noticeably, however these "improved" values where still very low as compared to the ideal value.

Trial	Frequency	Universal	Approximate Entropy
1	F	F	0.118532
2	P	F	0.084436
3	F	F	0.118543
4	P	F	0.084445
5	F	F	0.118544
6	F	F	0.118544
7	F	F	0.000015
8	P	F	0.000014
9	F	F	0.118543
10	F	F	0.118533

Table 2: Tests on the X-Generation X1 Generated Keystream

Trial	Frequency	Universal	Approximate Entropy
1	F	F	0.114675
2	F	F	0.107743
3	F	F	0.125585
4	F	F	0.107725
5	F	F	0.125577
6	F	F	0.114667
7	P	F	0.128362
8	F	F	0.129374
9	F	F	0.125615
10	F	F	0.114667

Table 3: Tests on the X-Generation X1 + MSK Generated Keystream

Tables 4 and 5 show the results of the experiments on the X2 X-Generation register. The pass rate is higher in the filter tests as compared to the high number of fails recorded in the X1 results. The entropy values are also higher than the X1 results. This illustrates the fact that the LFSR is longer (23 bits compared with the 6 bit length of X1) and has denser polynomials.

The pass rate is approximately the same for the X2/MSK sequence compared to the X2 sequence. Again the results are consistent and the application of MSK seems to lessen the strength of the original sequence in some cases.

Trial	Frequency	Universal	Approximate Entropy
1	P	F	0.674179
2	P	P	0.688198
3	P	P	0.688400
4	P	P	0.686629
5	P	P	0.688264
6	P	P	0.688641
7	P	P	0.686658
8	P	P	0.688208
9	P	P	0.688652
10	P	P	0.688380

Table 4: Tests on the X-Generation X2 Generated Keystream

Trial	Frequency	Universal	Approximate Entropy
1	P	P	0.687554
2	P	P	0.687888
3	P	P	0.687989
4	P	P	0.688181
5	P	P	0.687884
6	P	P	0.687855
7	P	P	0.687947
8	P	F	0.687798
9	P	P	0.688147
10	P	P	0.688076

Table 5: Tests on the X-Generation X2 + MSK Generated Keystream

Tables 6 and 7 show the results of the experiments on the X3 X-Generation register. Here the application of the MSK scheme seems to improve the sequence according to the Universal test criteria. The MSK scheme also improves the entropy values slightly. However this improvement is not significant when comparing the values to the ideal entropy value.

Trial	Frequency	Universal	Approximate Entropy
1	P	F	0.685090
2	P	F	0.685048
3	P	F	0.685000
4	P	F	0.685052
5	P	F	0.685053
6	P	F	0.685012
7	P	F	0.685055
8	P	F	0.685068
9	P	F	0.685023
10	P	F	0.685060

Table 6: Tests on the X-Generation X3 Generated Keystream

Trial	Frequency	Universal	Approximate Entropy
1	P	P	0.687508
2	P	F	0.687526
3	P	F	0.687503
4	P	P	0.687462
5	P	P	0.687426
6	P	P	0.687507
7	P	P	0.687579
8	P	P	0.687573
9	P	P	0.687549
10	P	F	0.687483

Table 7: Tests on the X-Generation X3 + MSK Generated Keystream

9.3 Applying MSK to the A5/1 Algorithm

Tables 8 and 9 show the results of the tests performed on the A5/1 generated stream and the A5/MSK generated keystream. Again one Mega bit files were generated. The results were consistent and show slightly more entropy overall when compared to the entropy results of the stronger X-generation registers. It can be seen that larger more complex registers improve the entropy. However this slight improvement would indicate a lack of sensitivity in tests performed as the A5/1 algorithm uses three registers with primitive polynomials used to generate a keystream in a non-linear fashion (refer section 3).

When comparing the two keystreams MSK does not seem to make any significant difference to the entropy values and in some cases seems to decrease the strength of the original A5 keystream.

Trial	Frequency	Universal	Approximate Entropy
1	P	P	0.688953
2	P	P	0.689020
3	P	P	0.689066
4	P	P	0.689067
5	P	P	0.689166
6	P	P	0.688983
7	P	P	0.689089
8	P	P	0.688993
9	P	P	0.689027
10	P	P	0.689081

Table 8: Tests on the A5/1 Generated Keystream

Trial	Frequency	Universal	Approximate Entropy
1	P	P	0.688931
2	P	P	0.689175
3	P	P	0.688888
4	P	P	0.689049
5	P	P	0.688925
6	P	P	0.688936
7	P	P	0.688978
8	P	P	0.689063
9	P	P	0.689216
10	P	P	0.689087

Table 9: Tests on the A5/1 + MSK Generated Keystream

10 Conclusions

MSK has been applied to a set of arbitrary weak keystream generators and to the A5/1 moderately strong keystream generator. It does not increase the entropy of keystreams nor does it resolve overall weaknesses which cause weak keystreams to fail basic statistical tests.

Figure 12 summarises the results of the tests performed on the A5/1 generated keystream and the MSK enhanced A5/1 generated keystream. The graph shows clearly that the MSK coding scheme does not make any significant difference to the randomness, and hence the strength, of the original keystream produced from the A5/1 stream cipher. In some cases the MSK scheme improves the entropy slightly and in others it actually reduces the entropy value. In the cases where the scheme improves the entropy, the resulting value is still low in comparison to the ideal entropy value of 0.693147.

From analysing these results it can be said that the author's claim that the MSK coding scheme "exponentially enhances" stream ciphers cannot be sustained.

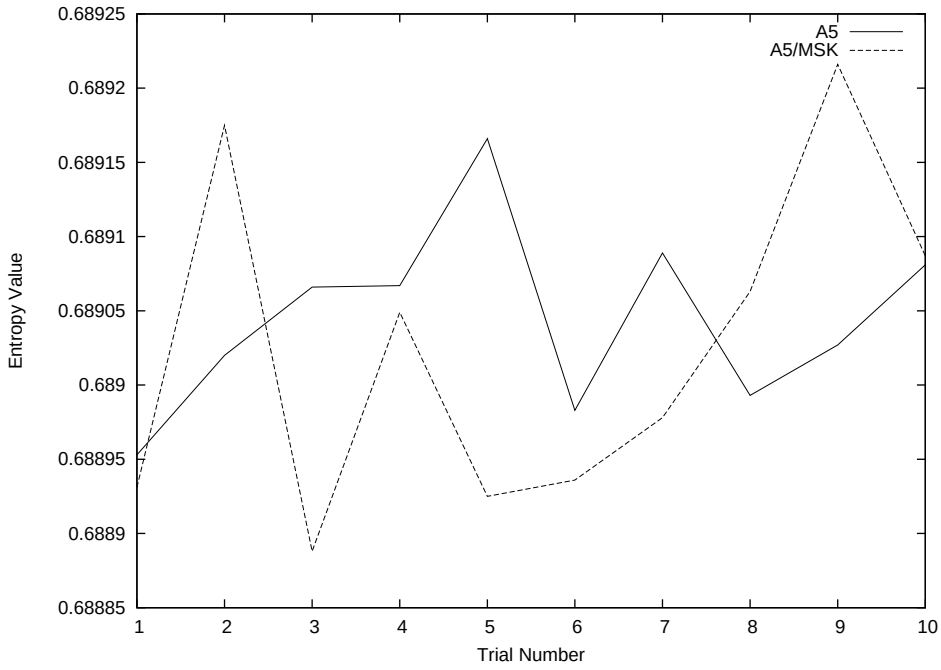

Figure 12: Comparison of A5 and A5/MSK keystreams

11 Future Work

As discussed for the purposes of this report three statistical tests were chosen for the analysis of the MSK coding scheme. However there many statistical tests discussed in literature for the testing of randomness of stream ciphers. The NIST SP 800-22 test suite includes other statistical tests that can be used for cryptographic keystream analysis. To strengthen the results found in this report further statistical tests could be performed to support the findings.

For the purpose of this report the values of M, S and K used where the the values suggested by the authors of the MSK coding scheme. Varying these parameters would provide for interesting future work to determine whether there is any difference in the keystream randomness when different values are used or even if one of these parameters where not used at all.

References

[1] David Wagner Alex Biryukov, Adi Shamir. *Real Time Cryptanalysis of A5/1 on a PC*. Fast Software Encryption Workshop 2000, April 2000.

[2] David Wagner Alex Biryukov, Adi Shamir. *Real Time Cryptanalysis of A5/1 on a PC*. http://cryptome.org/a51-bsw.htm, 2000.

[3] Campbell-Kelly Aspray, Bromley. *Computing before Computers*. Iowa State University Press, Ames, Iowa, 1990.

[4] Elaine B. Barker. *A Statistical Test Suite for Random and Pseudorandom number generators for Cryptographic Applications*. http://csrc.nist.gov/publications/, 2003.

[5] Cristian Calude. *The Definition of Random Strings*, volume 83. University of Auckland - Department of Computer Science, Nov 1993.

[6] Orr Dunkelman Eli Biham. *Cryptanalysis of the A5/1 GSM Stream Cipher*. Progress in Cryptology - INDOCRYPT 2000, India, Dec 2000.

[7] Eric Filiol. *Decimation Attack of Stream Ciphers*. Progress in Cryptology - INDOCRYPT, Calcutta, India, Dec 2000.

[8] Oded Goldreich. *Pseudorandomness*. Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, Israel, 1999. Technical report.

[9] J Golic. *Cryptanalysis of Alleged A5 Stream Cipher*. Proceedings of EUROCRYPT'97, LNCS 1233, 1997.

[10] W.H McAnney J Savir. *A multiple seed linear feedback shift register*. Test Conference, 1990 Proceedings, International, Sept 1990.

[11] K Jambunathan. *On Choice of Connection-polynomials for LFSR-based Stream Ciphers*. Progress in Cryptology - INDOCRYPT, Calcutta, India, Dec 2000.

[12] Keshariya A Keshariya M. *Development of Multiple Excess-S Modulo K (MSK) Codes and their application to Cryptography*. Proceedings of the national Seminar on Cryptography, Defence Research and Development Organisation, New Delhi, 1998.

[13] D Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, 1981.

[14] P. L. L'Ecuyer. *Uniform random number generation*. Annals of Operations Research, 1994.

[15] Helger Lipmaa. *A5 Stream Cipher*. http://www.tcs.hut.fi/ helger/crypto/link/stream/a5.html, 2003.

[16] David Margrave. *http://www.hackcanada.com*. GSM Security and Encryption, 2003.

[17] U. M. Maurer. *A Universal Statistical Test for Random Bit Generators*, volume 5. Journal of Cryptology, 1992.

[18] National Institute of Standards and Technology. *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*. NIST Special Publication 800-22, 2001.

[19] US National Institute of Standards and Technology. *Federal Information Processing Standard 140-2: Security Requirements for Cryptographic Modules*. NIST, 2001.

[20] T. Johansson P. Ekdahl. *Another attack on A5/1 [GSM stream cipher]*. 2001 IEEE International Symposium, 2001.

[21] Lauri Pesonen. *http://udsab.dia.unisa.it/ads.dir/corso-security*. GSM Interception, 1999.

[22] Graham Richter. *Design, Analysis, Implementation and Comparison of Stream Cipher Algorithms*. Department of Computer, Electrical and Electronic Engineering, University of Pretoria, 2002. Technical report.

[23] A. L. Rukhin. *The error probability, entropy, and equivocation when the number of input messages increases*, volume 42. Information Theory, IEEE Transactions on, Nov 1996.

[24] A. L. Rukhin. *Approximate Entropy for Testing Randomness*, volume 37. Journal of Applied Probability, 2000.

[25] S Micali S Goldwasser. *Probabilistic Encryption*, volume 28. Journal of Computer and System Sciences, 1984.

[26] H. Bauke S. Mertens. *Entropy of Pseudo Random Number Generators*. Institut fur Theoretische Physik, Otto-von-Guericke Universitat, Germany, May 2003.

[27] B. H. Singer S. Pincus. *Randomness and degrees of irregularity*, volume 93. Proceedings of the National Accademy of Sciences of the USA, 1996.

[28] R. E. Kalman S. Pincus. *Not all (possibly) "random" sequences are created equal*, volume 94. Proceedings of the National Accademy of Sciences of the USA, April 1997.

[29] B. Schneier. *Applied cryptography : protocols, algorithms, and source code in C*. J. Wiley and Sons, 1996.

[30] RSA Security. *http://www.rsasecurity.com/rsalabs/faq/2-1-5.html*. rsasecurity.com, 2003.

[31] Chris Thorn. *Randomness and Entropy - An Introduction*. SANS Institute 2003 - GSEC Practical v 1.4a, 2003.

[32] Securius Newsletter Vol 4.1 Security Through Usability. *http://www.securius.com/newsletters/Security-Through-Usability.html*. Securius.com, 2003.